STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 4, 2012

REDUNDANT SPATIAL INDEX FOR SOLVING RANGE QUERIES

LEON ŢÂMBULEA, ADRIAN SERGIU DĂRĂBANT,
AND ANDREEA NAVROSCHI-SZASZ

Abstract. Lately, new applications arise that manage large collections of location and points of interest objects. These are frequently accessed nowadays in mobile and web applications. In order to be able to query and update the position of these objects, several index structures are used. Each of these structures has its own advantages in solving a certain concrete problem. The term POI (Point of Interest) is employed in the context of use of mobile devices. A characteristic of POI collections of objects is their relatively static character (these objects do not usually change their location). In the index structures recommended for these collections a POI object is stored a single time. In this paper we suggest the alteration of an existing index structure by memorizing several times the addresses of some POI objects. The purpose of this multiple storage consists in reducing the response time for range queries.

Key-Words: Grid, R-tree, query POI objects, index redundancy

1. Introduction

A POI (Point of Interest) is a position or a complex 3D structure with some information associated with it (ID, civil address, category, name, description, etc.) [8].

The term of position (location) refers to a physical point on the Earth surface (specified within a system of coordinates). Some categories of POI objects are static (shopping center, speed camera, petrol station), but there are also POI objects valid only for a certain period of time (for example, a cultural event scheduled for a certain time interval).

Lately there are many applications that use spatial data queries, particularly POI objects collections. These applications start from a specified position

Received by the editors: November 28, 2012.

2000 *Mathematics Subject Classification.* 03G10.

1998 *CR Categories and Descriptors.* H.2.2 [**Database Management**]: Physical Design – *Access methods*; H.2.8 [**Database Management**]: Database applications – *Spatial databases and GIS*; H.2.4 [**Database Management**]: Systems – *query processing*.

or a position determined by a GPS device and they look for POI objects located near this position, with possible additional restrictions, returning the position and the associated information for the determined objects. For such applications it is necessary that the responses to these queries are obtained as quickly as possible. Large processing power and storage are generally not enough in order to achieve acceptable response times and high accuracy on this problem. One also needs specific types of indexes adapted to POI data and POI query processing. Among the index structures mostly used for this kind of queries we mention R-tree and the various versions of R-tree [2, 5, 9], the grid structures [1, 6, 7], etc. In each of these structures, a reference to an element from the POI objects collection is memorized a single time (there is no redundancy in the index).

In this paper we analyze the possibility of memorizing several times certain information (addresses of POI objects) in order to reduce the response time for these queries.

The rest of this paper is organized in the following way: section 2 shortly presents the R-tree and grid structures and the way to determine the responses to the queries, section 3 describes the suggested changes for the index associated to a grid, in section 4 we emphasize the results obtained for some experiments, and in section 5 we formulate some conclusions.

2. R-tree and Grid Structures

The POI objects collection is stored into a data base. Each object has an *idPOI*, a *position(x,y)*, and other associated information.

A range query reference is specified by two opposite points in a rectangle D. With this type of query we ask for all POI objects located inside the rectangle D.

To avoid the sequential access to all the objects of the data base, we build an index. For each POI object in the index we memorize at least ($idPOI$ and (x, y)).

2.1. **R-tree.**

The R-tree structure [2] was widely studied and used. An R-tree is a balanced tree with two types of nodes:

- end nodes, where we store a sequence of values $(idPOI, (x, y))$;
- internal nodes, where we store a sequence of values $(idchild, MBR)$. *idchild* is a pointer to an internal or end node, and MBR (minimum bounding rectangle) is the smallest rectangle that includes all the objects that arise in the sub-trees of this node. There is no criterion for the order of elements from an internal or end node.

As parameters of the index we have [4]:

- the dimension of a node of the R-tree, from which we can determine the maximum number of inputs to the node: the number of $(idchild, MBR)$ pairs from an internal node, or the number of $(idPOI, (x, y))$ values from an end node. Let M be the maximum number of entries that will fit in one node.
- the minimum number of entries in one node (which is not the root node) is given by a second parameter $m \leq M/2$. This parameter dictates the minimum occupation of a node.

An internal node is analyzed (the nodes referred by this one are covered) if the associated MBR intersects the rectangle D. For a terminal node all the $(idPOI, (x, y))$ values are examined and the objects for which $(x, y) \in D$ are included in the query answer. Depending on the query, it's possible that several branches of the tree need to be covered.

2.2. **Grid.** The domain where the POI objects are located is divided into a regular network of square cells (resulting into a matrix of cells) (see Figure 1). The objects located inside such a cell are attached to this one and are stored into a list. For a more efficient browsing of the list, the objects can be stored into a linked list of blocks (a block may contain a specified maximum number of $(idPOI, (x, y))$ elements).

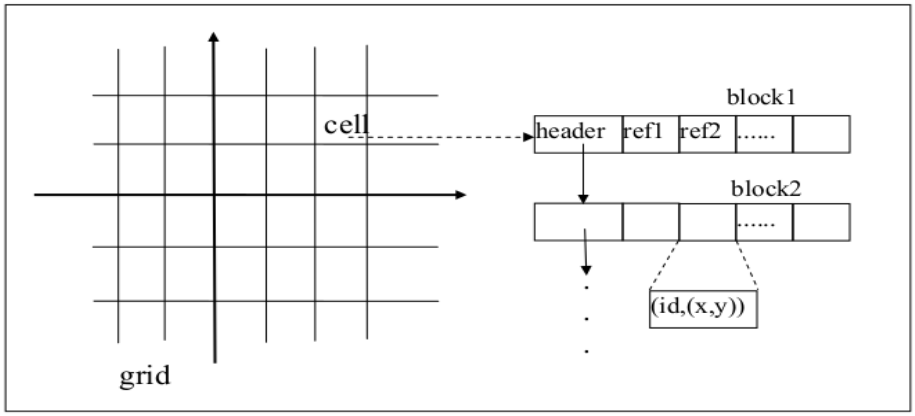

FIGURE 1. Grid index structure

The parameters of this index are [10]:

- The dimension used for dividing the domain of POI objects (the side of a square cell from the grid);
- The dimension of a block.

To determine the answer to a range query the following two steps need to be taken:

(a) We determine the cells that are completely included inside the rectangle D (all the elements from these cells are included in the answer);
(b) We determine the cells that are partially covered by the rectangle D. For the objects of these cells we perform the test $(x, y) \in D$.

The POI objects are not uniformly spread in the plane, therefore some cells will have many blocs (of objects) associated, and the search from step b) can include a lot of data. In [3], the authors propose the use of a several levels grid (the cells with many objects are organized as new grids).

Another possible index consists in storing the list of POI objects that belong to a certain cell into a R-tree. Therefore the index structure is made of a collection of R-trees and a matrix of references to these R-trees.

3. Redundant Index

Let d be the dimension of a cell from the grid built for the collection of POI objects (according to section 2.2). We assume that the rectangle D from the range query is a square and it corresponds to the following query: find all POI objects located within a maximum distance d_0 from a point (x_0, y_0). To increase the efficiency of the search, we first find the POI objects located on the coordinate axes (on each direction), within a maximum distance d_0 from point (x_0, y_0), therefore:

$D(x_0, y_0; d_0) = \{(x, y) \in \Re \times \Re |\ |x - x_0| \leq d_0, |y - y_0| \leq d_0\}$

Let s be the maximum possible value of d_0. If s is large, then:

- the number of cells from the grid queried is large (according to section 2.2);
- the answer to the range query specified by $D(x_0, y_0, d_0)$ contains a large number of recordings, situation that is detrimental for most applications that use such queries.

Next we suggest a possible alteration of the index structure if $s < d/2$, and in the next section we prove that in some cases the response time for range queries respecting the above conditions is reduced.

We build a "main index" for the grid, as suggested in Section 2.2. For the above hypothesis ($s < d/2, d_0 \leq s$), the answer to a range query is located in one, two or four cells of the grid, as can be seen in Figure 2.

Let q be a range query, $D(x_0, y_0, d_0)$ the associated square and $z(x_0, y_0)$ the cell in the grid where point (x_0, y_0) appears. The answer to the query q is found by browsing:

- the objects associated to the cell $z(x_0, y_0)$. Let $n(q)$ the number of objects in this cell;

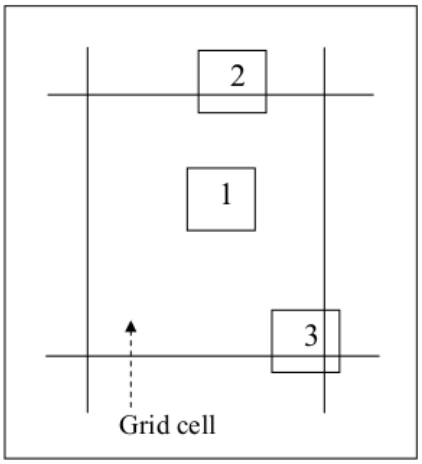

FIGURE 2. Possible overlays of the range query answer over the grid area

- the objects from the neighboring cells that intersect D. Let $n_s(q)$ be the number of objects in these cells. If D is in the first case stated by Figure 2, then $n_s(q) = 0$.

We will denote by $n_1(q)$ the number of objects queried in order to find the answer to q, so $n_1(q) = n(q) + n_s(q)$.

An additional index - "the secondary index" is added to the grid. It contains, for a given cell z, references to objects in the neighboring cells (those that share a side with z) and located within a minimum distance s from the sides of the cell z. The areas containing the additional objects attached to a cell are highlighted in Figure 3.

With this version of index, the answer to a query q specified by $D(x_0, y_0, d_0)$ is found by:

- browsing the $n(q)$ objects from the main index associated to the cell z;
- browsing the objects in the secondary index associated to the cell z, if D is in case 2 or 3 stated by Figure 2. Let $n_{si}(q)$ be the number of objects in the secondary index that need to be queried. If $D \subseteq z$, then $n_{si}(q) = 0$.

We will denote by $n_2(q)$ the number of objects queried in order to find the answer to q, therefore $n_2(q) = n(q) + n_{si}(q)$.

Let C be the collection of POI objects. For a certain d (the dimension of a cell from the grid) and s (the maximum dimension for the d_0 values from the

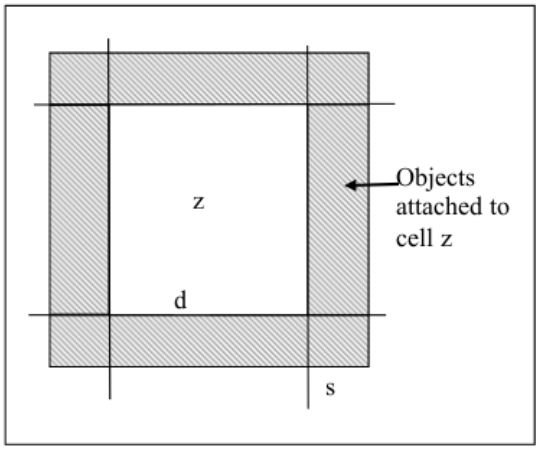

FIGURE 3. Redundant objects attached to a grid cell in the secondary index

range query) we can build the two index structures (the main and secondary one) by a single traversal of collection C.

For an object $ob = (id, (x, y)) \in C$ we determine:

(a) $(i, j) = (\lfloor \frac{x}{d} \rfloor, \lfloor \frac{y}{d} \rfloor)$ and the object is included in the main index of cell (i, j);

(b) $if\ x \in I_1 = [i \cdot d, i \cdot d + s)\ then\ left := i - 1\ else\ left := null$;
$if\ x \in I_2 = [(i+1) \cdot d - s, (i+1) \cdot d)\ then\ right := i + 1\ else\ right := null$;
$if\ y \in J_1 = [j \cdot d, j \cdot d + s)\ then\ bottom := j - 1\ else\ bottom := null$;
$if\ y \in J_2 = [(j+1) \cdot d - s, (j+1) \cdot d)\ then\ top := j + 1\ else\ top := null$;

(c) If at least one of the previous four conditions is satisfied then object ob is included in the secondary index of cell (m, n). We denote by $S_i(ob; m, n)$ the operation of inserting ob in the secondary index.

```
if (left!= null) and (top!=null) then Si(ob; left , top);
if (top!=null) then Si(obj; i , top);
if (right!=null) and (top!=null) then Si(ob; right , top);
if (right!=null) then Si(ob; right , j);
if (right!=null) and (bottom!=null) then Si(ob; right , bottom);
if (bottom!=null) then Si(ob; i , bottom);
if (left!=null) and (bottom!=null) then Si(ob; left , bottom);
if (left!=null) then Si(ob; left , j);
```

Figure 4 illustrates the above presented cases. It can be seen that:

- each object is included in the main index for a single cell;
- objects from the subareas z_2, z_4, z_6, z_8 are contained in a single cell of the secondary index;
- objects from subareas z_1, z_3, z_5, z_7 are contained in three cells of the secondary index;

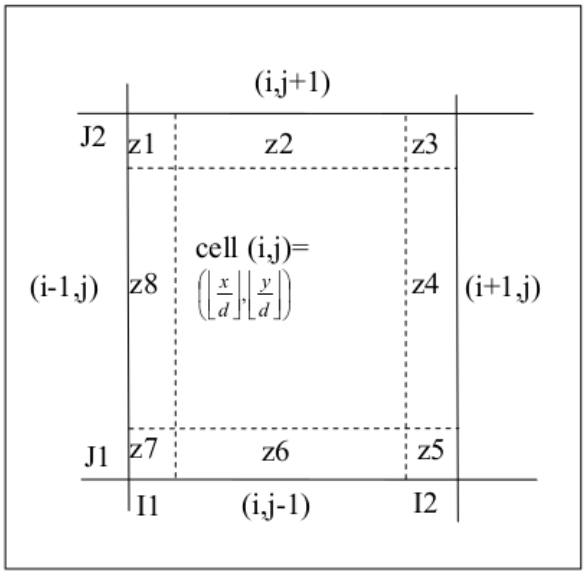

FIGURE 4. Objects included in the secondary index.

By redundantly storing references to some objects, the index size grows, with the size of the secondary index. The induced redundancy coefficient can be expressed by the following formula:

$$c = \frac{size(secondary\_index)}{size(main\_index)} \tag{1}$$

This coefficient depends on the d and s system parameters and on the object distribution in the generated grid.

For a set of range queries Q we can determine if using the main and a secondary index is more efficient than the main only index version. In order

to achieve this we introduce the efficiency coefficient:

$$e = \frac{\sum_{q \in Q} n_2(q)}{\sum_{q \in Q} n_1(q)} \tag{2}$$

For $e < 1$, the total number of accessed objects when applying the new indexing schema for all queries in Q is smaller than in a classical grid structure index.

4. Experimental results

In this section we present the experimental results obtained when varying the values of the parameters introduced in the previous sections.

4.1. **The Object Database Collection.** In the conducted experiments we used a real POI database collection provided by one of the major actors in this field on the market. The main characteristics of this collection are:

- the number of POI objects is 1,195,398;
- the area covered by the database collection is $latitude \in [-54.8, 78.2], longitude \in [-179.8, 179.4]$.

For experiments we generated 100,000 range queries, each query being expressed by a location and a range $(x_0, y_0; d_0)$. The (x_0, y_0) locations were randomly generated on the area covered by the POI collection. For each given s, the d_0 parameter values were randomly generated in the interval $[s \div 4, s]$.

4.2. **Measurements.** For each d we compute the number of non-empty cells in the grid. Table 1 shows the number of non-empty cells for a few values of the d parameter.

d(km)	50	100	150	200	300
Cell Count	11,412	5,243	3,268	2,295	1,384

Table 1. Number of non-empty cells vs grid size.

For a given d and s (a subset of the experimental values) and for all queries in Q Table 2 shows:

- *Secondary Index Size* - the size of the secondary index;
- c - the redundancy coefficient;
- $N(q) = \sum_{q \in Q} n(q)$;

- *Additionally inspected cells* - the number of additional grid cells intersected by all queries;
- $N_s(Q) = \sum_{q \in Q} n_s(q)$;
- $N_{si}(Q) = \sum_{q \in Q} n_{si}(q)$;
- the efficiency coefficient $e = \frac{N(Q)+N_{si}(Q)}{N(Q)+N_s(Q)}$;

From the Table 2 it can be seen that the efficiency coefficient has a value less than 1 for many situations. In all these cases the number of analyzed objects for a variant with a secondary index is smaller than the number of analyzed objects for a main grid index without redundancy.

Also, the redundancy coefficient has in these cases reasonable values. This means the overhead incurred for the multiple storage of the same objects would not impact significantly the data structures stored in the main memory. By

d (km)	s	Secondary Index Size	c	N(Q)	Additionally inspected cells	$N_S(Q)$	$N_{SI}(Q)$	e
50	5	470,919	0.42	481,095	24,907	103,381	40,642	0.893
50	10	985,209	0.88	481,095	50,078	242,636	170,561	0.900
50	15	1,600,183	1.43	481,095	74,781	293,400	385,180	1.119
100	5	227,479	0.20	1,853,072	12,468	221,971	52,538	0.918
100	10	456,930	0.41	1,853,072	25,061	513,589	176,288	0.857
100	20	1,017,484	0.91	1,853,072	50,403	880,128	682,995	0.928
100	30	1,670,481	1.50	1,853,072	75,182	1,248,760	1,616,484	1.119
150	5	165,392	0.15	4,206,558	8,299	276,158	45,101	0.948
150	10	336,591	0.30	4,206,558	16,630	676,243	187,649	0.900
150	15	510,677	0.46	4,206,558	24,844	930,088	446,989	0.906
150	20	693,832	0.62	4,206,558	33,276	1,249,643	837,634	0.924
150	30	1,110,691	1.00	4,206,558	49,852	1,721,729	1,745,026	1.004
200	5	111,985	0.10	7,320,215	6,232	610,148	68,648	0.932
200	10	222,985	0.20	7,320,215	12,615	1,153,923	214,695	0.889
200	20	511,359	0.46	7,320,215	25,107	1,953,250	797,491	0.875
200	30	800,983	0.72	7,320,215	37,600	2,768,881	1,697,463	0.894
200	40	1,091,544	0.98	7,320,215	49,984	3,578,632	3,188,074	0.964
200	50	1,389,982	1.25	7,320,215	62,557	3,955,536	4,818,800	1.077
300	5	81,718	0.07	17,126,951	4,187	579,144	51,908	0.970
300	10	158,232	0.14	17,126,951	8,347	1,201,525	186,371	0.945
300	20	331,249	0.30	17,126,951	16,612	2,522,389	768,913	0.911
300	30	533,765	0.48	17,126,951	24,796	3,617,474	1,857,947	0.915

TABLE 2. Number of non-empty cells vs grid size.

analyzing these two factors over a set of queries Q and a typical database, one can decide whether adding a secondary redundant index like the one proposed above can improve or not the query response time.

5. Conclusions

In this paper we proposed an improvement on the indexing of a statical collection of POI objects by adding an additional index. The additional index uses only redundant references that are copies of some of the entries from the main grid index. In some cases the proposed method improves the response time by 10%-15% compared to a typical R-tree or Grid index at the cost of some additional required memory. Update operations are considered in the same context as for classical R-trees and Grid indexes and are usually expensive. This is a well known problem of the R-tree index as well. The conducted experiments show that there are real life situations where applying the proposed method helps reducing query response time at the cost of very little added memory.

References

[1] J. L. Bentley, J. H. Friedman, *Data Structures for Range Searching*, ACM Comput. Surv., 11, 4, 397409, 1979.

[2] A. Guttman, *R-Trees - A Dynamic Index Structure for Spatial Searching*, SIGMOD Rec., 14(2):4757, 1984.

[3] D. V. Kalashnikov, S. Prabhakar, S. E. Hambrusch, *Main Memory Evaluation of Monitoring Queries Over Moving Objects*, Distrib. Parallel Databases, 15, 2, 2004, 117-135.

[4] Ling. Liu, M. Tamer zsu (Eds.), *Encyclopedia of Database Systems*, Springer, 2009.

[5] Y. Manolopoulos, A. Nanopoulos, A.N. Papadopoulos, Y. Theodoridis, *R-Trees: Theory and Applications*, Series in Advanced Information and Knowledge Processing, Springer 2005.

[6] Ming Qi, Guangzhong Sun, Yun Xu, *Query as Region Partition in Managing Moving Objects for Concurrent Continuous Query*, International Journal of Research in Computer Science, 2 (1): pp. 1-6, 2011.

[7] J. Nievergelt, H. Hinterberger, K. C. Sevcik, *The Grid File: An Adaptable, Symmetric Multikey File Structure*, ACM Transactions on Database Systems, 9(1):3871, 1984.

[8] *Points of Interest Core*, W3C Working Draft 12 May 2011, http://www.w3.org/TR/poi-core/.

[9] A.Sabau, *Management of Spatio-Temporal Databases*, PhD Thesis, Cluj-Napoca 2007.

[10] D. Sidlauskas , S. Saltenis , Ch. W. Christiansen , J. M. Johansen , D. Saulys, *Trees or Grids? Indexing Moving Objects in Main Memory*, Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2009, Seattle, Washington.

Babes Bolyai University, Faculty of Mathematics and Computer Science, Cluj Napoca, Romania

E-mail address: leon@cs.ubbcluj.ro, dadi@cs.ubbcluj.ro, deiush@cs.ubbcluj.ro